43장 Ajax

43.1 Ajax란?

- **Ajax(Asynchronous JavaScript And XML)**란 자바스크립트를 사용하여 브라우저가 서버에게

  비동기 방식으로 데이터를 요청하고 수신하는 프로그래밍 방식

- Ajax는 브라우저에서 제공하는 Web API인 XMLHttpRequest 객체를 기반으로 동작
- XMLHttpRequest는 HTTP 비동기 통신을 위한 메서드와 프로퍼티를 제공
- 이전의 웹페이지는 화면이 전환될 때마다 서버로부터 새로운 HTML을 전송받아 웹페이지 전체를 처음부터 다시 렌더링
  - 이러한 전통적인 방식은 다음과 같은 단점이 있다.
    1. 변경 사항이 없는 파일도 매번 다시 전송받기 때문에 불필요한 데이터 통신이 발생한다.
    2. 변경할 필요가 없는 부분도 다시 렌더링되어 화면 전환 도중 깜박이는 현상이 발생한다.
    3. 통신이 동기적으로 이루어지기 때문에 블로킹 현상이 일어난다.
  - Ajax가 등장하면서 이러한 전통적인 방식의 단점이 해결될 수 있게 되었다.
    1. 변경할 부분을 갱신하는 데 필요한 데이터만 전송받아 불필요한 데이터 통신이 발생하지 않는다.
    2. 변경할 필요가 없는 부분은 다시 렌더링되지 않기 때문에 화면이 깜박이는 현상이 발생하지 않는다.
    3. 통신이 비동기적으로 이루어지기 때문에 블로킹이 발생하지 않는다.
       - 즉, 변경 사항이 있는 데이터만 비동기 방식으로 전송받아 불필요한 렌더링이

         일어나지 않고 빠른 퍼포먼스와 부드러운 화면 전환이 가능해졌다.

43.2 JSON

- **JSON(JavaScript Object Notation)**은 클라이언트와 서버 간의 HTTP 통신을 위한 문자열 데이터 형식
  - 자바스크립트에 종속되지 않아 대부분의 프로그래밍 언어에서 사용할 수 있다.

43.2.1 JSON 표기 방식

- JSON은 자바스크립트의 객체 리터럴과 유사하게 키와 값으로 구성된 순수한 텍스트
  - JSON의 키와 문자열 값은 반드시 큰따옴표(작은따옴표 x)로 묶어야 한다.

```
{
    "name": "Lee",
  "age": 20,
  "alive": true,
  "hobby": ["traveling", "tennis"]
}
```

43.2.2 JSON.stringify

- 객체나 배열을 JSON 포맷의 문자열로 변환
- **직렬화** : 클라이언트가 서버로 객체를 전송하기 위해 객체를 문자열화

1. 객체 → JSON 변환

```
const obj = {
    "name": "Lee",
    "age": 20,
    "alive": true
};

// 객체를 JSON 포맷의 문자열로 변환한다.
const json = JSON.stringify(obj);
console.log(typeof json, json);
// string {"name": "Lee","age": 20,"alive": true,"hobby":
["traveling", "tennis"]}

// 객체를 JSON 포맷의 문자열로 변환하면서 들여쓰기 한다.
const prettyJson = JSON.stringify(obj, null, 2);
```

```
console.log(typeof prettyJson, prettyJson);
/*
string {
    "name": "Lee",
    "age": 20,
    "alive": true,
    "hobby": [
        "traveling",
        "tennis"
    ]
}
*/
```

2. 배열 → JSON 변환

```
const todos = [
    {id: 1, content: 'HTML', completed: false},
  {id: 2, content: 'CSS', completed: true},
  {id: 3, content: 'JavaScript', completed: false},
];

const json = JSON.stringify(todos, null, 2);
console.log(typeof json, json);

/*
string [
    {
        id: 1,
        content: 'HTML',
        completed: false
    },
    {
        id: 2,
        content: 'CSS',
        completed: true
    },
    {
        id: 3,
```

```
        content: 'JavaScript',
        completed: false
    },
]
*/
```

43.2.3 JSON.parse

- JSON 형식의 문자열을 객체나 배열로 변환
- 서버로부터 클라이언트에 전송된 JSON 데이터는 문자열
- **역직렬화** : 문자열을 객체로서 사용하기 위해 JSON 포맷의 문자열을 객체화

1. JSON → 객체 변환

```
const obj = {
    "name": "Lee",
  "age": 20,
  "alive": true,
  "hobby": ["traveling", "tennis"]
};

// 객체를 JSON 형식의 문자열로 변환한다.
const json = JSON.stringify(obj);

// JSON 형식의 문자열을 객체로 변환한다.
const parsed = JSON.parse(json);
```

2. JSON → 배열 변환

```
const todos = [
    {id: 1, content: 'HTML', completed: false},
  {id: 2, content: 'CSS', completed: true},
  {id: 3, content: 'JavaScript', completed: false},
];

// 배열을 JSON 포맷의 문자열로 변환한다.
const json = JSON.stringify(todos, null, 2);
```

```
// JSON 포맷의 문자열을 배열로 변환한다. 배열의 요소까지 객체로 변환한다.
const pared = JSON.parse(json);
```

43.3 XMLHttpRequest

- 자바스크립트에서 HTTP 요청을 전송하기 위한 객체
- Web API인 XMLHttpRequest 객체는 HTTP 요청 전송과 HTTP 응답 수신을 위한 다양한 메서드 및 프로퍼티 제공

43.3.1 XMLHttpRequest 객체 생성

```
    const xhr = new XMLHttpRequest();
```

43.3.3 HTTP 요청 전송

```
// XMLHttpRequest 객체 생성
const xhr = new XMLHttpRequest();

// 1. HTTP 요청 초기화
xhr.open('GET', '/user');

// 2. HTTP 요청 헤더 설정
// 클라이언트가 서버로 전송할 데이터의 MIME 타입 지정: json
xhr.setRequestHeader('content-type', 'application/json');

// 3. HTTP 요청 전송
xhr.sennd();
```

43.3.4 HTTP 응답 처리

```
// ✅ JSONPlaceholder에서 제공하는 가상 REST API 사용

// XMLHttpRequest 객체 생성
const xhr = new XMLHttpRequest();

// HTTP 요청 초기화
// https://jsonplaceholder.typicode.com은 Fake Rest API를 제공하
xhr.open('GET', 'https://jsonplaceholder.typicode.com/todos/1

// HTTP 요청 전송
xhr.sennd();

// readystatechange 이벤트는 HTTP 요청의 현재 상태를 나타내는 readyS
// 변경될 때마다 발생한다.
xhr.onreadystatechange = () => {
    // readyState 프로퍼티는 HTTP 요청의 현재 상태를 나타낸다.
    // readyState 프로퍼티 값이 4(XMLHttpRequest.DONE)가 아니면
    // 서버 응답이 완료되지 않는 상태다.
    // 만약 서버 응답이 아직 완료되지 않았다면 아무런 처리를 하지 않는다.
    if (xhr.readyState !== XMLHttpRequest.DONE) return;

    // status 프로퍼티는 응답 상태 코드를 나타낸다.
    // status 프로퍼티 값이 200이면 정상적으로 응답된 상태이고
    // status 프로퍼티 값이 200이 아니면 에러가 발생한 상태다.
    // 정상적으로 응답된 상태라면 response 프로퍼티에 서버의 응답 결과가
    if (xhr.status === 200) {
        console.log(JSON.parse(xhr.response));
        // {userId: 1, id: 1, title: "delectus aut autem", co
    } else {
        console.error('Error', xhr.status, xhr.statusText);
    }
};

// load 이벤트는 HTTP 요청이 성공적으로 완료된 경우 발생한다.
xhr.onload = () => {
    if (xhr.status === 200) {
    console.log(JSON.parse(xhr.response));
    // {userId: 1, id: 1, title: "delectus aut autem", comple
```

```
    } else {
        console.error('Error', xhr.status, xhr.statusText);
    }
};
```